Mobile Application Programming

Views
class Car
{
    private var _vin: String
    private var _year: Int
    private var _position: Point // Imported
    private var _velocity: Vector // Imported

    init(vin: String, year: Int)
    {
        _vin = vin
        _year = year
        _position = Point(x: 0.0, y: 0.0)
        _velocity = Vector(x: 0.0, y: 0.0)
    }

    var vin: String
    {
        return _vin
    }

    var year: Int
    {
        return _year
    }

    var position: Point
    {
        get { return _position }
        set { _position = newValue }
    }

    var velocity: Vector
    {
        get { return _velocity }
        set { _velocity = newValue }
    }

    func move(elapsedTime: Double)
    {
        _position += _velocity * elapsedTime
    }
}

// Usage
var viper: Car = Car("23958060934985", 2003)
viper.position = Point(x: 40.76, y: -113.93)
viper.velocity = Vector(x: 100.0, y: 200.0)
viper.move(1.2)
Views

- **UIView** instances and subclasses
- Form a tree rooted at the window
- Have a backing store of pixels that are drawn seldomly, then **composited** to form the full user interface
- Coordinate system for drawing uses **bounds** property
- Positioned within parent using **center** or **frame** property
- Subclasses typically override **drawRect** method
- Receive touches overriding super-class methods **touchesBegan**, **touchesMoved**, and **TouchesEnded**
Graphics Architecture

Views / Layers
- Drawing

Graphics Commands

Hardware
- Compositing

iJezzball

New Game
High Scores
How to Play
Sound ON
Graphics Context

- C API around CGContext usually obtained from an image or calling UIGraphicsGetCurrentContext()
- Device-Independent rendering by using transform matrices
- Contains an associated graphics state (actually a stack of graphics states maintained by push / pop)
- Similar to Java Graphics2D
Painter’s Algorithm

Quartz 2D Programming Guide
Anti-Aliasing

Quartz 2D Programming Guide
Paths
Paths - Lines

```swift
func CGContextMoveToPoint(c: CGContext!, x: CGFloat, y: CGFloat)

func CGContextAddLineToPoint(c: CGContext!, x: CGFloat, y: CGFloat)
```
```swift
func CGContextAddQuadCurveToPoint ( c: CGContext!, cpx: CGFloat, cpy: CGFloat, x: CGFloat, y: CGFloat )
```

**Paths - Curves**
func CGContextAddCurveTo Alright (. c: CGContext!, cp1x: CGFloat, cp1y: CGFloat, cp2x: CGFloat, cp2y: CGFloat, x: CGFloat, y: CGFloat)
Paths - Rectangles

```swift
func CGContextAddRect ( c: CGContext!, rect: CGRect )
```

func CGContextAddEllipseInRect (c: CGContext!,
rect: CGRect)

func CGContextAddArc (c: CGContext!, x: CGFloat, y: CGFloat, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat, clockwise: Int)
Rounded Rectangles
func CGContextDrawPath (c: CGContext!, mode: CGContextDrawingMode)

kCGPathFill,
kCGPathEOFill,
kCGPathStroke,
kCGPathFillStroke,
kCGPathEOFillStroke
Stroke Properties

`func CGContextSetLineWidth(c: CGContext, width: CGFloat)`

`CGContextSetLineCap`

<table>
<thead>
<tr>
<th>Style</th>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butt cap</td>
<td></td>
</tr>
<tr>
<td>Round cap</td>
<td></td>
</tr>
<tr>
<td>Projecting square cap</td>
<td></td>
</tr>
</tbody>
</table>

`CGContextSetLineJoin`

<table>
<thead>
<tr>
<th>Style</th>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miter join</td>
<td></td>
</tr>
<tr>
<td>Round join</td>
<td></td>
</tr>
<tr>
<td>Bevel join</td>
<td></td>
</tr>
</tbody>
</table>

`CGContextSetLineDash`
func CGContextClip (c: CGContext!)
Gradients

```swift
func CGContextDrawLinearGradient (c: CGContext!,
    gradient: CGGradient!,
    startPoint: CGPoint,
    endPoint: CGPoint,
    options: CGGradientDrawingOptions)
```
func CGContextSaveGState (c: CGContext!);

func CGContextRestoreGState (c: CGContext!)

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current transformation matrix (CTM)</td>
</tr>
<tr>
<td>Clipping area</td>
</tr>
<tr>
<td>Line: width, join, cap, dash, miter limit</td>
</tr>
<tr>
<td>Accuracy of curve estimation (flatness)</td>
</tr>
<tr>
<td>Anti-aliasing setting</td>
</tr>
<tr>
<td>Color: fill and stroke settings</td>
</tr>
<tr>
<td>Alpha value (transparency)</td>
</tr>
<tr>
<td>Rendering intent</td>
</tr>
<tr>
<td>Color space: fill and stroke settings</td>
</tr>
<tr>
<td>Text: font, font size, character spacing, text drawing mode</td>
</tr>
<tr>
<td>Blend mode</td>
</tr>
</tbody>
</table>
Drawing Images and Strings

- CoreGraphics API exists for these, but UIKit is easier

```swift
var someImage: UIImage = UIImage(named: "image.png")
someImage.drawInRect(self.bounds)

var someString: String = String(format: "Size: %i", arguments: 5)
var systemFont: UIFont = UIFont.systemFontOfSize(18.0)
someString.drawInRect(self.bounds, withAttributes:
    [NSFontAttributeName: systemFont])
```
UIBezierPath

- **Object-oriented** way to create and draw paths
- Build up paths using `moveToPoint`, `addLineToPoint`, etc. or use `constructors` for ellipses and rectangles
- Uses the “current context” automatically when calling the methods `stroke` or `fill`
- Stroke properties accessible through `lineWidth`, etc.
- Use `setStroke` and `setFill` methods on `UIColor` to set colors prior to stroke and fill calls
Coordinate Systems

Properties
- Frame
- Bounds
- Center
- Transform
Coordinate Systems

Frame: (5, 5, 73, 98)  
Bounds: (0, 0, 73, 98)  
Center: (39, 50.5)

Frame: (5, 5, 73, 98)  
Bounds: (8, 24, 73, 88)  
Center: (39, 50.5)
Transform

\[
\begin{bmatrix}
  a & c & t_x \\
  b & d & t_y \\
  0 & 0 & 1
\end{bmatrix}
\]

CGAffineTransformMake(a, b, c, d, tx, ty)

Applied to Bounds, Relative to Center
func convertPoint(p: CGPoint toView v: UIView)
func convertPoint(p: CGPoint fromView v: UIView)
func convertRect(r: CGRect toView v: UIView)
func convertRect(r: CGRect fromView v: UIView)
PaintCode

www.paintcodeapp.com
Example: Knob
Example: Knob

knobView.bounds
knobRect
nibRect
angle